Multipath transport and packet spraying for efficient data delivery in data centres

Article (Accepted Version)

Multipath Transport and Packet Spraying for Efficient Data Delivery in Data Centres

Morteza Kheirkhah[a], Ian Wakeman[b], George Parisis[b]

[a]*Department of Electronic & Electrical Engineering, University College London, UK*
[b]*School of Engineering and Informatics, University of Sussex, UK*

Abstract

Modern data centres provide large aggregate network capacity and multiple paths among servers. Traffic in data centres is very diverse; most of the data is produced by long, bandwidth hungry flows but the large majority of flows, which commonly come with stringent deadlines regarding their completion time, are short. It has been shown that TCP is not efficient for any of these types of traffic in modern data centres. MultiPath TCP (MPTCP) employs multipath data transport and is efficient for long flows but ill-suited for short flows.

In this paper, we present Maximum MultiPath TCP (MMPTCP), a novel transport protocol which extends MPTCP and, compared to TCP and MPTCP, reduces short flows' completion times, while providing excellent goodput to long flows. To do so, MMPTCP runs in two phases; initially, it randomly scatters packets in the network under a single congestion window exploiting all available paths. This is beneficial to latency-sensitive flows. After a specific amount of data is sent, MMPTCP switches to a regular MultiPath TCP mode. MMPTCP is incrementally deployable in existing data centres as it does not require any modifications outside the transport layer and behaves well when competing with MPTCP flows. We also present a topology-specific extension of MMPTCP that adjusts the numbers of subflows during the second phase of the protocol based on knowledge about the location of the receiver in the data centre.

We present extensive evaluation that shows that MMPTCP's design objectives are met. We have implemented MMPTCP (along with MPTCP and packet spraying) in ns-3 and evaluated our protocol in simulated FatTree topologies. We have evaluated how MMPTCP performs compared to TCP and MPTCP and how its performance is affected by transient hotspots in the network. We have also experimented with different thresholds for duplicate acknowledgements and fast retransmissions and shown that MMPTCP performs well when the size of short flows is widely ranged. Finally, we have evaluated how MMPTCP performs under conditions that result in Incast, when different congestion control algorithms are used in its second phase and when varying the overall network load.

Keywords: Data Center Network, Multipath Transport, Congestion Control, Flow Scheduling

1. Introduction

Modern data centre network architectures [1, 2, 3] provide very high aggregate bandwidth and dense interconnectivity in the network by incorporating multiple paths among servers. They support a large number of network services which produce very diverse intra-data centre traffic matrices. The majority of the data is produced by long flows, which are bandwidth-hungry. Short flows commonly come with stringent deadlines regarding their completion time. According to [2], "99% of flows are smaller than 100 MB, however, more than 90% of bytes are in flows between 100 MB and 1 GB". If short flows cannot deliver all their data before their deadlines, some data may be discarded, decreasing the overall quality of the main computation or forcing some tasks to be restarted, wasting CPU and network resources. Deadlines are typically missed due to encountering transient or persistent congestion in their paths. Short flows result in very bursty and unpredictable traffic patterns, which in turn means that data centres are susceptible to severe transient congestion in any link in the network. Web latency, part of which can be induced within the data centre, is inversely correlated with revenue and profit; Amazon estimates that every 100ms increase in latency cuts profits by 1% [4]. In [5], it stated that "a 500 millisecond delay in the Bing search engine reduced revenue per user by 1.2%, or 4.3% with a 2-second delay".

Equal-Cost Multi-Path (ECMP) routing [6] is nowadays deployed in data centre switches so that multiple paths can be used to efficiently route data in the network. However, even with ECMP in place, TCP is ill-suited for both long and short flows within the data centre. Under high load, long flows collide with high probability and, as a result, network utilisation significantly drops and only 10% of the flows achieve their expected throughput [7]. TCP is also inefficient for short flows, especially when competing with long flows. Queue build-ups, buffer pressure and TCP Incast combined with the shared-memory nature of data centre switches results in short TCP flows often missing their deadlines mainly due to retransmission timeouts

Email addresses: m.kheirkhah@ucl.ac.uk (Morteza Kheirkhah), Ianw@sussex.ac.uk (Ian Wakeman), g.parisis@sussex.ac.uk (George Parisis)

(RTOs) [8].

Several transport protocols have been recently proposed to deal with these challenges. DCTCP [8], D2TCP,[9] and D3 [10] all aim at reducing flow completion times (FCT) for latency-sensitive flows. However, they require modifications in the network and/or deadline-awareness at the application layer. Such information may not be known a priori (i.e. at connection time). Worse, these protocols are not designed to co-exist with other transport protocols, and thus have a problematic deployment path.

Multipath transport protocols, such as MultiPath TCP (MPTCP) [11], transfer data using multiple subflows and rely on ECMP to distribute the subflows to several network paths. As shown in [7], MPTCP achieves high goodput and improves the overall network utilisation. This is also illustrated in Figure 1(a)[1], where MPTCP with eight subflows almost doubles the application goodput when compared to TCP (i.e. MPTCP with a single subflow in Figure 1(a)). However, MPTCP handles short flows inefficiently. The congestion window of a subflow may be very small over its lifetime. As a result, even a single lost packet can force an entire connection to wait for an RTO to be triggered because this lost packet cannot be recovered through fast retransmission. This is clearly illustrated in Figure 1(b), where the mean short flow completion time increases as more subflows are used (better shown in the embedded Figure). Note that the number of connections that experience one or more RTOs significantly increases as well, hence the increase in the standard deviation. Note that even a single RTO may result in flow deadline violation.

Central flow scheduling approaches, such as Hedera [13], primarily deal with long flows. Hedera detects long TCP flows at the edge switches and its central controller schedules these to optimise bandwidth allocation. Short flows are a secondary consideration, and their completion times suffer from the TCP pathologies described above.

Supporting and running multiple transport protocols in a data centre can be problematic. Fairness among different protocols is difficult to achieve; most protocols for latency-sensitive flows are not compatible with TCP or MPTCP [8, 9]. Running multiple transportppn protocols is also a burden for application developers who would have to decide upon the most suitable transport protocol. Both application requirements and data centre topologies evolve over time and so a transport protocol that performs well over disparate topologies and traffic matrices is a necessity.

In this paper, we present MMPTCP [14, 15], a multipath transport protocol that extends MPTCP and aims at:

1. high throughput for long flows;
2. low latency for short flows;
3. tolerance to sudden and high bursts of traffic;
4. minimal changes to the network architecture;
5. fair co-existence with legacy transport protocols.

[1]For these simulations we used our custom implementation of MPTCP in ns-3 [12]. Our source code is publicly available via a GitHub repository that can be found in https://github.com/mkheirkhah/mmptcp

MMPTCP achieves its objectives by transferring data in two phases. Initially, it randomly scatters packets in the network under a single congestion window exploiting all available paths. This is beneficial to latency-sensitive flows, which typically have bursty traffic patterns. After a specific amount of data is sent, MMPTCP switches to a regular MultiPath TCP mode, efficiently handling long flows through separate congestion windows for each subflow.

The remainder of this paper is as follows: in section 2, we present the design of the proposed transport protocol and its influences from Packet Scatter [16] and MPTCP [7]. We describe the problems associated with scattering packets in the network and packet reordering, when multiple paths are used, and discuss our proposed solution. We also present a topology-specific extension of MMPTCP that adjusts the numbers of subflows during the second phase of the protocol based on knowledge about the location of the receiver in the data centre. Section 3 presents our extensive evaluation of MMPTCP in simulated data centre topologies. Simulations are based on our MMPTCP implementation in ns-3.We have used synthetic traffic matrices and realistic data centre application workloads, as described in [17]. Section 4 explores potential future improvements of MMPTCP with respect to the congestion control algorithm used during the first phase of our protocol, multi-homed data centre topologies and QoS features that are available in modern data centres.

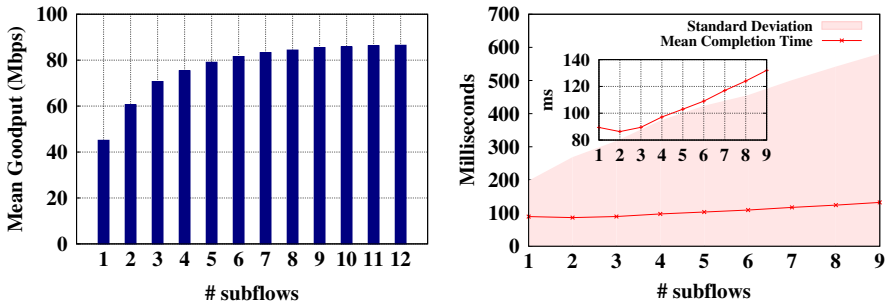
2. Design

In this section, we discuss the Packet Scatter (PS) and MPTCP protocols before describing MMPTCP, which has been designed based on these two protocols. We also discuss spurious retransmissions due to packet reordering, which are a key challenge for MMPTCP, and describe our solution which is embedded in the proposed protocol. Finally, we describe a topology-specific extension of MMPTCP that exploits knowledge about the network topology and location of servers to set the number of subflows that are used in its second phase.

2.1. Packet Scatter

Data transport through scattering (spraying) packets in the data centre network has been briefly explored in [7] and discussed in more details in [16]. The key idea behind Packet Scatter (PS) is that ECMP-enabled network switches choose one of the valid output ports on a per-packet instead of a per-flow basis, as in Valiant Load Balancing [18]. Traffic can thus be distributed as evenly as possible among all paths between two endpoints. The corollary of packets within a flow taking multiple paths is that packet reordering becomes more likely and so the protocol must use more robust Fast Retransmit algorithms to deal with out-of-order packets.

It has been argued that if traffic load is equal among servers and a data centre has a uniform network topology, such as FatTree [1] or VL2 [2], then PS achieves perfect load balancing in the network core and eliminates congestion from that layer [7]. However, although traffic that is switched on a per-packet



(a) Goodput (b) Flow Completion Time (FCT)

Figure 1: Goodput for long flows (1(a)) and completion time for short flows (1(b)) in a full-bisection and 4:1 over-subscribed FatTree topology consisting of 128 and 512 servers, respectively. One third of the servers run long (background) flows. The rest run short flows (70KBs each) whose arrival follows a Poisson process). All flows are scheduled based on a permutation traffic matrix (see Section 3.1).

basis does not create hotspots at the network core, traffic that is distributed through ECMP on a per-flow basis (e.g. TCP or MPTCP flows) may still end up sharing links and causing congestion. Network hardware failures or traffic from the Internet to the data centre, which typically consists of regular TCP flows, may also cause such congestion [7]. Since PS flows share a single congestion window, if a packet is dropped, then the congestion window shrinks across all paths that the flow is using, drastically reducing its throughput. Further, the first congestion event across the paths used by the PS flow is governed by the probability distribution of the *minimum* number of packets per path before loss across the different paths, reducing the expected time before window reduction[2]. Finally, if PS flows compete with regular TCP flows in a shared bottleneck link at the network access layer, then network fairness is preserved because both transport protocols use the TCP congestion control algorithm.

2.2. MultiPath TCP

MultiPath TCP is an extension of TCP that transfers data through multiple paths simultaneously. It actively senses network congestion for all its subflows and shifts traffic from more to less congested paths within a few RTTs. Unlike TCP, MPTCP deals with network congestion gracefully by putting fewer packets to the congested subflows. The main requirement to achieve such behaviour is to retain a congestion window for each subflow and link each of them together. This is the main reason why MPTCP (with eight subflows) doubles the mean goodput of long flows, compared to TCP, as illustrated in Figure 1(a). Furthermore, MPTCP flows can co-exist with TCP flows gracefully, fairly sharing the network resources with them [7].

MPTCP is an appealing approach for data centres where traffic patterns are highly diverse and volatile. However, MPTCP is susceptible to retransmission timeouts when a MPTCP flow contains only a few packets, because it uses multiple congestion windows (one for each subflow). When packets get dropped from a subflow of a very small MPTCP flow, this subflow is most likely unable to use the fast retransmission mechanism to recover the lost packets. As a result, even a single packet drop from a single subflow could hold an entire MPTCP connection back until the lost packet has been recovered through the retransmission timeout mechanism, although all other subflows have already completed their data delivery.

As the number of subflows in small MPTCP flows increases, the likelihood of relying on retransmission timeouts for recovering lost packets increases. As illustrated in Figure 1, increasing the number of subflows is beneficial to the goodput of long flows, but it is harmful to short flows' completion time.

The question that is raised is whether it is possible to adjust the number of MPTCP subflows based on the size of the flow. It has been argued that some applications can provide high-level information, such as flow size [10], to the transport layer. If such information was available, one could decide how many subflows it might be effective to use. For example, in the case of short flows, it is better to have a single subflow. Unfortunately, the majority of applications do not expose their flow sizes to the transport layer[3]. If a predefined number of subflows is used for all types of flows then MPTCP is likely to significantly hurt short flows' completion time. Note that even if MPTCP can use a single subflow for short flows, it still cannot prevent transient congestion in the network, which mainly occurs due to the bursty nature of short flows. MMPTCP is designed to prevent transient congestion and to effectively mitigate persistent congestion in data centre networks (see Section 2.3).

2.3. MMPTCP: Combining PS with MPTCP

Before delving into the mechanics of MMPTCP, we briefly enumerate its main design principles and objectives:

1. Handle short flows through packet scatter, mitigating MPTCP's short flow inefficiencies.
2. Handle long flows through standard MPTCP.

[2] If we grossly simplify the likelihood of a packet loss on a path to the geometric distribution with parameter p, then the expected minimum across n paths is $1/(1-(1-p)^n$.

[3] It would be possible to get information about the size of a flow when using the *sendfile()* system call through a short-lived transport layer connection. However, this cannot be considered the norm.

3. Decrease the burstiness of data centre networks, which mainly originates from short flows, by diffusing packets throughout the network to reduce transient congestion.
4. Exploit topology-specific information (e.g. for topologies like FatTree and VL2) to minimise spurious packet retransmissions and minimise the number of subflows in the network without sacrificing flows' perceived goodput.

The core idea behind MMPTCP is that, initially, data is transferred by scattering packets in the network until the amount of transmitted data reaches a certain threshold. To do so, we employ source port randomisation at the source host and standard ECMP at network switches. Note that MPTCP allows the usage of different source ports for subflows when this can potentially provide disjoint paths through ECMP [11]. A token[4] is added to each packet of the initial subflow as a connection identifier so that sprayed packets can be forwarded to the corresponding MMPTCP connections.[5] The standard 5-tuple connection identification is no longer valid during the initial phase of data transmission because of the source port randomisation. Transport is governed by a single congestion window throughout the duration of the first phase, whose aim is to take advantage of all available network paths and quickly complete short flows. The usage of the token ensures that there can be no clash between packets with randomised source ports (that are part of MMPTCP's first phase) and packets that are exchanged during the second phase of an MMPTCP connection, in the presence of multiple connections between two servers in the data centre.

When the switching threshold is reached, MMPTCP switches to standard MPTCP with multiple subflows to benefit from MPTCP's efficiency in dealing with long flows. The initial subflow is only allowed to scatter packets in the network during the first phase; after switching to MPTCP, no more packets are put in the initial subflow, which is deactivated when its window gets emptied. We explicitly deactivate the first subflow because there may still be unacknowledged segments from the first phase, after switching to the second phase. These segments may get acknowledged after all MPTCP subflows are opened, as part of the second phase. One could try to reset the first subflow to be a regular MPTCP flow, if it was emptied of data before all MPTCP subflows were opened in the second phase - however this would not affect the performance of MMPTCP. During the second phase, data transmission is governed by MPTCP's congestion control mechanism.

In the initial handshake of MPTCP, SACK may also be activated if DSACK is used as a part of the packet reordering strategy (see Section 2.4). MPTCP is compatible with SACK, so there is no problem in having SACK activated over the lifetime of an MMPTCP connection. On the other hand, DSACK would only be used in the first phase to minimise spurious retransmissions due to out-of-order packets.

[4] A token is a locally unique identifier assigned to a MMPTCP connection upon establishment.

[5] Every packet during the initial phase of MMPTCP (i.e. the packet scattering phase) carries a token (a 32-bit unique MPTCP connection identifier). As a result, the loss of one packet would not produce any negative effect in MMPTCP's operation.

2.4. MMPTCP and Packet Reordering

A TCP sender may receive a duplicate acknowledgement (duplicate ACK) when a packet gets dropped, delayed or reordered. It enters the Fast Retransmit phase upon the arrival of the third duplicate ACK for a missing packet (when the duplicate ACK threshold parameter is set to three). It retransmits the perceived lost packet and halves its congestion window as a reaction to the congestion signal. However, the Fast Retransmit mechanism may still be falsely triggered and a reordered packet can reach the receiver after the host has sent a third duplicate ACK. This condition may lead to spurious retransmissions of reordered packets even if no loss has occurred; i.e. the sender misinterprets the reordered packet as lost. As a result, the sender falsely triggers the Fast Retransmit mechanism and halves its congestion window, which, in turn, leads to performance degradation.

Although this condition is unlikely to occur with TCP/MPTCP flows in data centres (whose packets take the same path throughout the flow lifetime), it is common when scattering packets in the network, since RTTs on different network paths may vary over time due to queuing delays. MMPTCP must therefore handle packet reordering during its first phase in order to be able to meet its objectives with respect to completion times of short flows.

Setting the right *dupthresh* value is not trivial; if *dupthresh* is too low, spurious retransmissions become the norm. If it is too high, the sender may react to congestion through a retransmission timeout instead of the Fast Retransmit mechanism; obviously this would be a very undesirable situation as even a single timeout may lead a flow to miss its deadline. Our experimental evaluation in Section 3.5 confirms these observations.

There are three key aspects in making TCP more robust to packet-reordering: *preventing*, *detecting* and *mitigating* spurious retransmissions due to out-of-order packets.

One well-known solution for detecting and mitigating spurious retransmissions is DSACK [19], which is an extension of SACK TCP [20]. SACK TCP can deal with multiple packet drops much faster than other versions of TCP (e.g. NewReno [21]). This is particularly beneficial for latency-sensitive flows. When a spurious retransmission is detected by DSACK, the state of the congestion window can be simply reversed to the state when a loss is detected.

One possible approach for preventing spurious retransmissions is to dynamically adjust the *dupthresh* parameter based on information that can be retrieved from DSACK, SACKs, ACKs, RTOs and Fast Retransmits. RR-TCP [22] follows a similar approach; it attempts to adjust the *dupthresh* value dynamically by inferring the maximum distance in packets by which a segment is displaced, using the mechanisms described above.

Our approach for preventing out-of-order packets is to set the value of *dupthresh* based on topology-specific information. For example, FatTree's IP addressing scheme can be exploited to calculate the number of available paths between a sender and a receiver. The sender can thus choose an appropriate value for the *dupthresh* based on this information. For example, if a source sends its traffic via the core layer, then the *dupthresh*

should be much higher, compared to when traffic crosses only a Top-Of-the-Rack (TOR) switch.

In this paper we use FatTree's addressing scheme as the basis for setting *dupthresh*. Each source host detects the layer(s) of the network topology that its traffic would cross when transmitting data to a specific destination host, by examining the destination IP addresses. For example, when a connection needs to be established between hosts with IP addresses *10.0.1.1* and *10.0.1.2* then, based on FatTree's addressing scheme, both hosts are located within the same ToR switch; therefore the *dupthresh* value should not be changed from the default value of three. Traffic crossing the aggregate or core layers would require higher *dupthresh* values. In such scenarios, we propose to calculate the *dupthresh* value by adding the number of equal-cost paths that are available to a flow to the default *dupthresh* value.

The knowledge of the end-host's location is essential but not sufficient to assign an appropriate value for the *dupthresh*; each end-host also needs to know the size of the network topology. A network topology with 4 core switches requires a different value of *dupthresh* compared to a network topology with 8 core switches. Additionally, network switches may also support ECMP with a limited number of paths in each IP subnet (e.g. up to 16 equal-cost paths), therefore knowing these information is also important for deciding a precise value for duplicate ACK threshold. For example, if a flow crosses the core layer in a FatTree topology with 16 core switches, the value of *dupthresh* is 19 since the packets of this flow can be delivered via 16 distinct paths to its destination. In Section 3.5, our experimental evaluation confirms that this approach significantly decreases spurious retransmissions. Our approach could be easily incorporated to other topologies, such as VL2 [2]. The VL2 agent in combination with the directory system's resolution service could provide all necessary information about the location of recipients and the size of the network to senders, so that the *dupthresh* value can be adjusted accordingly.

Another option could be to dynamically adjust the dupack threshold based on the observed events and congestion signals, as in the Linux TCP implementation. Incorporating such a dynamic approach in the first phase of MMPTCP would be problematic; short flows are usually completed within a few RTTs and therefore the number of observable events to adjust the threshold towards the optimal value would potentially be insufficient. In such cases, starting from a value that is far from the optimal one would severely hurt the performance of short flows due to either a large number of spurious retransmissions (when packet reordering in the network is high and the dupack threshold is low) or slow response to losses (when a loss occurs and the dupack threshold is high).

Note that as the over-subscription ratio decreases and the provision of full bisection bandwidth becomes the norm (as in modern data centres), packet losses at the network core and aggregation layers become very rare (e.g. 0.0001 (intra-pod) to 0.00001 (inter-pod) [23]). In such network environments, setting the *dupthresh* value to a large value is not a problem since packets get rarely dropped. However, if a network exhibits higher loss rates, then setting a large value for *dupthresh* may increase the number of timeouts because the Fast Retransmit mechanism cannot be triggered (see Section 3.10 for demonstration of this problem). While our approach for dealing with out-of-order packets neither increases the *dupthresh* value to a very large value nor for all network flows, to be conservative, we recommend to activate the TCP Limited Transmit (LT) mechanism on the initial phase of MMPTCP when a network exhibits high packet drop probability. LT is an enhancement to TCP loss recovery which attempts to prevent RTOs, especially when the congestion window size is very small [24, 22]. LT allows a TCP sender to transmit new segments upon arrival of only the first two duplicate ACKs for a specific segment, i.e. before the fast retransmission is triggered. We employ a modified version of this algorithm so that a TCP sender can send new segments before fast retransmission is triggered, regardless of the *dupthresh* value. For example, if *dupthresh* is 19 then a TCP sender can send 18 new segments before triggering the fast retransmission. This way, a sender can prevent timeouts when a packet gets dropped while the congestion window is smaller than *dupthresh*. Section 3.6 demonstrates the performance improvement of MMPTCP with LT. In our experimental evaluation, we run MMPTCP without enabling the LT mechanism, unless otherwise stated.

2.5. Dynamically Setting the Number of Subflows

In MMPTCP's second phase, a sender opens multiple subflows to the receiver and distributes data through all of them. The key objective in opening multiple subflows is to compensate for ECMP collisions when two flows are routed through the same network path while other, equal-cost paths could be used. If there are no equal-cost paths between a pair of communicating hosts, then having multiple subflows only wastes CPU and memory resources on both sides and potentially affects fairness of resources' utilisation with respect to competing flows. If there is a small number of equal-cost paths between a pair of communicating hosts, then running a large number of subflows is also inefficient because most of these will collide with each other. Following the rationale for setting the *dupthresh* value, we utilise a topology-specific approach for managing the number of sub-flows that are opened after switching from the packet scattering phase. If the receiver is located within the same rack as the sender, then only one sub-flow is used. Depending on the topology and the location of the communicating hosts (i.e. whether traffic from the sender to the receiver crosses aggregation and core switches) more equal-cost paths are available, therefore more subflows are opened. In Section 10, we evaluate this scheme (which we call 'Auto Subflow') and demonstrate that MMPTCP with 'Auto Subflow' reduces the FCT of short flows at the tail, compared to standard MMPTCP. In our evaluation, we run MMPTCP without enabling the 'Auto Subflow' mechanism, unless otherwise stated.

3. Evaluation

Simulating large data centre networks is challenging because of their large size. Increasing the size of the network (network switches and servers), the number of senders and receivers,

the traffic workload and the available link rates results in dramatically increasing the time it takes to complete simulations. We have found that with ns-3 we could realistically conduct all required simulations for this paper if we kept link rates to 100Mbps. For the same reason we have used a 8-ary FatTree topology. Only for the first set of experiments (Section 3.2), where we compare how MMPTCP performs in comparison to MPTCP, we have used link rates of 1Gbps. Based on the experience gathered by running sample (and short) simulations using faster supported link rates, there would be no significant differences to the results reported in this paper, if we conducted our simulations with 1Gbps links. On the contrary, MMPTCP performs better compared to the other studied protocols when faster links (e.g 1Gbps) are simulated.

The MPTCP model presented in [12] follows the MPTCP standard and does not use any extensions, such as SACK, DSACK, except Limited Transmit in Section 3.6. As in Linux, we use the loss recovery part of the TCP NewReno. The actual multipath congestion control algorithm is Linked Increase, which only modifies the 'increase' part of the standard TCP congestion control algorithm (i.e. TCP NewReno). A full window of data is sent through the initial subflow, before more subflows are opened. We allow some time to elapse before creating each one of these subflows; this time is uniformly selected at random between 0 and 50 microseconds. Unless otherwise stated, experimentation has been conducted using the following parameters: dupack threshold: 3, RTO: 200ms, initial congestion window: 1 MSS, #subflows for MPTCP and MMPTCP: 8, ssthresh: 65KBs, MSS: 1400 bytes, MMPTCP's switching threshold: 100KBs.

We have fixed the per-link propagation delay to 20us, the maximum queue size at each switch port to 100 packets, and the Maximum Transfer Unit (MTU) to 1500 bytes.

The results for each presented experiment in this paper are retrieved by averaging the results of 20 simulations with different seeds. Simulations typically take around 15 hours (wall clock time) to complete for a 4:1 oversubscribed, 8-ary FatTree topology (with 512 servers) with link rates of 100Mbps[6].

3.1. Traffic Matrices and Flow Scheduling

In the experiments presented in this paper we have used three different traffic matrices, which have been shown to be effective in revealing the complex behaviour of data centre networks [7][13].

Stride Matrix. Whenever a connection has to be established, a source host with id X connects to a destination host with id Y, where $Y = (X + I)\ mod\ (\#servers)$ and I is a matrix-specific parameter; I must be such that a server only connects to servers outside its own pod. The construction of the Stride traffic matrix follows three rules: (1) each server has only one incoming and one outgoing connection, (2) all flows are routed through the core layer, and (3) the source and destination hosts for each flow are located in different pods. By following these rules, if traffic is distributed evenly between servers and ECMP[7] uniformly distributes flows between equal-cost paths, then each server is expected to transmit data at its line rate. This model is particularly useful to study how network resources are shared among flows. For example, if the network flows are hypothetically distributed as evenly as possible throughout the core layer, the total throughput of all flows should be in line with the aggregate capacity of the core layer. With this traffic matrix we can examine network protocols in full bisection bandwidth topologies, because the matrix can saturate the network core.

Permutation Matrix. A source host is connected to a randomly selected destination host that does not have any incoming connections. Each host can only have one incoming and outgoing connection and there is no guarantee that a flow will be crossing the core layer. Compared to the Stride matrix, with the Permutation matrix the network core is less loaded.

Random Matrix. Source and destination hosts are randomly selected. Additionally, each host can have multiple incoming and outgoing connections.

For most simulations we have used a mixture of short and long flows and the traffic matrices described above. A fraction of servers exchange data continuously over the course of the simulation, in order to provide enough background traffic to congest the network core. The number of servers in this subset is determined by the amount of traffic necessary to saturate the network core. For example, in order to saturate the network core of a 4:1 FatTree topology when the Stride traffic matrix is used, 25% of servers need to run background flows. For the Permutation matrix this percentage value should be higher (e.g. 33%) as not all flows traverse the core layer. The remainder of the servers establish connections for short flows only. Each simulation scenario is named after the transport protocols used for short and long flows. For example, if a simulation scenario uses MPTCP for long flows and TCP for short flows, we refer to it as $\text{MPTCP}_{\text{TCP}}$. If a simulation scenario uses a single protocol for both long and short flows, we refer to it with the name of the protocol (e.g. MPTCP).

Short flows are scheduled by a central flow scheduler and their arrivals follow a Poisson process with mean arrival rate of λ flows per second. For each new flow, the central scheduler picks a random source server from a pool of servers that are used for short flows. The selected source server establishes a connection to its destination host and sends 70KBs of data[8]. Note that the central scheduler does not deal with how subflows and scattered packets are allocated to paths in the network. This is done using ECMP.

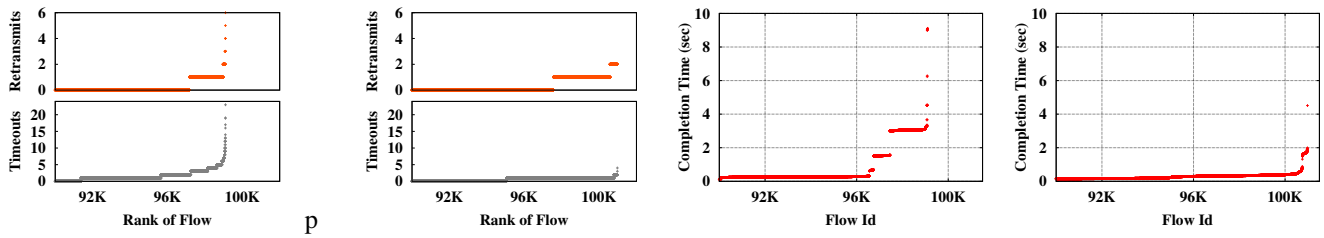
3.2. MMPTCP vs MPTCP

In Figure 1 we showed that although MPTCP performs well with respect to long flows' goodput, it is inefficient when it

[6] A brief discussion on the influence of the topology on the protocol behaviour can be found in [7]. Experimenting with different data centre topologies is left for future work.

[7] We have implemented ECMP in ns-3 so that paths are determined by hashing the source IP, source port, destination IP, destination port, and the protocol field in each packet; i.e. as specified in [25].

[8] 70KBs is the default flow size used in most of our experiments. In section 3.7 we simulate scenarios where flow sizes are drawn from a uniform distribution and a real production data centre workload [17].



(a) MPTCP with eight subflows (b) MMPTCP (c) MPTCP with eight subflows (d) MMPTCP

Figure 2: MPTCP with eight subflows vs MMPTCP: timeouts, fast retransmissions (2(a) and 2(b)) and short flow completion times (2(c) and 2(d)).

comes to short flows' completion times. If the amount of data on a subflow is very small (because of the small overall size of the flow) then even a single packet drop may lead to a retransmission timeout because there are not enough packets (and hence ACKs) to trigger fast retransmission. In other words, a single packet drop from a subflow can hold the entire MPTCP connection until that packet is recovered given that there is a single window for the whole connection.

Link Rates (Mbps)	Transport Protocol	Short Flow Completion Time (mean/stdev)	Long Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
100	**MPTCP**	126/±425 ms	62.1/±19.7 Mbps	75.5 %	0.0077 %
100	**MMPTCP**	116/±101 ms	61.9/±20.0 Mbps	74.9 %	0.0076 %
1000	**MPTCP**	38.5/±288.5 ms	718.1/±172.6 Mbps	70.5 %	0.0025 %
1000	**MMPTCP**	16.1/±54.7 ms	719.7/±168.5 Mbps	70.5 %	0.0025 %

Table 1: MPTCP with eight subflows vs MMPTCP

We have simulated a 4:1 oversubscribed FatTree topology consisting of 512 servers to compare MPTCP with eight subflows and MMPTCP. One third of the servers run long (background) flows. The rest run short flows (70KBs each), whose arrival is determined by a Poisson process with $\lambda = 256$ and the Permutation traffic matrix [7, 26]. MMPTCP's switching threshold is 100KBs [8, 27, 28]. The results are depicted in Table 1. The average short flow completion time for MPTCP running in a FatTree with link rates of 100Mbps is 126ms and the respective standard deviation is 425ms for 99103 completed short flows. The high standard deviation indicates that there are many cases in which MPTCP performs far worse than the average. The average flow completion time for MMPTCP is 116ms and the respective standard deviation is 101ms for a total of 100980 completed short flows. This is a significant improvement which means that MMPTCP short flows maintain their ACK clock better than MPTCP with eight subflows when they experience packet loss. This is because MMPTCP holds a single congestion window at the initial packet scattering phase of data delivery.

Using the same simulation setup but increasing the simulated link rates to 1Gbps, MMPTCP achieves more than 50% lower average FCT for short flows compared to MPTCP; this improvement is 5 times larger than when using 100Mbps links (approximately 10% as shown in Table 1). MMPTCP performs better with faster links; with faster links, network queues, which get full under bursty traffic loads resulting in transient congestion, get emptied very quickly (i.e. 10 times faster when comparing 1Gbps vs 100Mbps). As a result, short flows may experience less queueing delay. Moreover, for the same reason the variation of queue sizes is smaller when faster links are used, therefore packet reordering is less frequent. Both of these result in smaller short flow completion times.

Figures 2(a) and 2(b) illustrate the flow completion times, total number of fast retransmissions and timeouts of each individual short flow for MPTCP and MMPTCP, respectively, when the link rates are 100Mbps. It is clear that MPTCP suffers from excessive timeouts. Note that a few short flows experienced more than 20 timeouts and around ~4000 short flows experienced more than two timeouts during their lifetime. MMPTCP clearly outperforms MPTCP; it decreases the maximum number of timeouts and fast retransmissions from 25 to 4 and 6 to 2 respectively. The majority of short flows (more than 100000) experienced fewer than two timeouts (~95000 flows did not experience any timeout). Figure 2(c) and 2(d) depict the short flow completion times for MPTCP and MMPTCP, respectively. It is expected that with MPTCP a lot more short flows experience very high completion times due to the larger number of timeouts compared to MMPTCP.

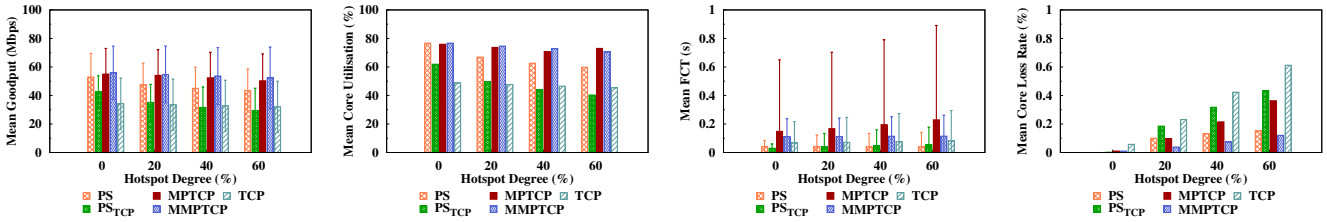
So far, we have shown that, unlike MPTCP, MMPTCP does not produce a heavy tail of short flow completion times, while it achieves high overall network utilisation and exceptional goodput for long flows. MMPTCP can therefore be deployed in existing data centres and used with all existing applications without relying on application information regarding flow sizes and potential deadlines. This is particularly important for data centre application designers who prefer not to consider underlying networking protocols when developing their applications.

Transport Protocol	Short Flow Completion Time (mean/stdev)	Long Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
PS	36.9/±38 ms	58.6/±18.2 Mbps	75.1 %	0.0001 %
TCP	64.3/±118 ms	38.5/±19.8 Mbps	44.7 %	0.0259 %
MMPTCP	116/±101 ms	61.9/±20.0 Mbps	74.9 %	0.0076 %
MPTCP	126/±425 ms	62.1/±19.7 Mbps	75.5 %	0.0077 %

Table 2: MMPTCP vs TCP and PS ($\lambda = 256$)

3.3. MMPTCP vs TCP and PS

In this section we compare the performance of MMPTCP, TCP and PS using the same simulation setup as the one presented in section 3.2. For each scenario, both short and long flows are served by the same protocol. The results are depicted in Table 2. TCP achieves the worst overall core



(a) Goodput of long flows (b) Utilisation of the core layer (c) FCT of short flows (d) Loss rate of the core layer

Figure 3: Performance evaluation for various hotspot degrees (0% to 60% of core switches). MMPTCP achieves the lowest loss rate, the highest mean core utilisation and goodput of long flows for all hotspot degrees compared to all other protocols.

utilisation and highest mean core loss rate. However, its mean flow completion time is lower than MMPTCP and MPTCP. PS achieves the lowest average short flow completion time and overall core loss rate with a high average goodput for long flows.

The key to understand these rather unintuitive results is the fact the both short and long flows are transported by the same protocol, and no competition between different protocols exists. TCP performs badly with respect to network resources' utilisation because it transports data through a single path, therefore it is unable to shift traffic to least congested paths (because it is a single-path transport protocol). TCP gets trapped in a congested path and damages itself and other competing flows at bottleneck links along the path. This is the very reason why TCP achieves lower short flow completion times (which initially seems unintuitive) compared to MPTCP or MMPTCP; unused capacity in the network is used by short flows to complete their data delivery quickly. In other words, the inability of long TCP flows to utilise all available network resources provides headroom for short TCP flows to be completed faster. PS performs well in this experiment because it prevents the creation of any congestion in the core and aggregation layers by scattering packets of all flows in the network.

After this analysis, one might question the benefits of running MPTCP and/or TCP in today's data centres, if PS can perform that well (as shown above). An important question here is why PS did not achieve the highest average goodput for long flows, even though we observed very low loss rate in the network core. As discussed in Section 2.1, PS supports a single congestion window and, as a result, when a loss is detected, the rate of data transmission is halved. Unlike MPTCP, PS has no way to shift traffic to the least congested paths [7]. If PS coexisted with other transport protocols, such as TCP and/or MPTCP, its performance would be significantly degraded. To examine this argument, we rerun the simulations above with λ = 2560 instead of 256 [7]. This simulation is used to study how the congestion control of each transport protocol behaves under highly dynamic traffic patterns (and heavier load) but also explains how the congestion is dealt with by each transport protocol in the network. Furthermore, we designed a simulation, referred to as PS_{TCP}, where short flows are transported by TCP and long flows by PS. With this scenario we evaluate the performance of PS when it competes with non-PS flows, such as TCP. The results are presented in Table 3.

For the PS_{TCP} scenario, we observe the lowest mean flow completion time (where short flows are handled by TCP). The average goodput for long flows (PS) is around 13Mbps lower than the maximum observed average (when both short and long flows are transported by MMPTCP). The average core utilisation is also 15% less compared to MMPTCP. This is because long flows in the PS_{TCP} scenario are more susceptible to packet loss, and hence they reduce their rates more frequently. When a buffer gets filled up, their packets most likely are in the tail of the queue since long flows randomly spread their packets via all possible paths. This lower network utilisation helps short flows (TCP in this case) to complete their data delivery without experiencing any collision and with less queuing delays. It is evident that PS is very sensitive to network congestion and when it is used for handling long flows it hurts their perceived goodput, and consequently the overall network utilisation. PS would not be able to compete with TCP or MPTCP because of the single congestion window maintained by each flow. TCP (for both short and long flows) achieves the least average goodput and the highest loss rates in the network core compared to all other protocols. MMPTCP achieves a lower mean flow completion time and standard deviation compared to MPTCP with eight subflows. It also achieves the same overall network utilisation as MPTCP.

Transport Protocol	Short Flow Completion Time (mean/stdev)	Long Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
PS	40.5/±44.3 ms	52.9/±16.7 Mbps	76.8 %	0.0001 %
PS_{TCP}	29.7/±31.1 ms	42.5/±11.3 Mbps	61.9 %	0.0014 %
TCP	66.5/±150 ms	34.2/±18.1 Mbps	48.8 %	0.0576 %
MMPTCP	111/±127 ms	55.9/±18.7 Mbps	76.7 %	0.0105 %
MPTCP	148/±502 ms	55.0/±18.2 Mbps	75.9 %	0.0100 %

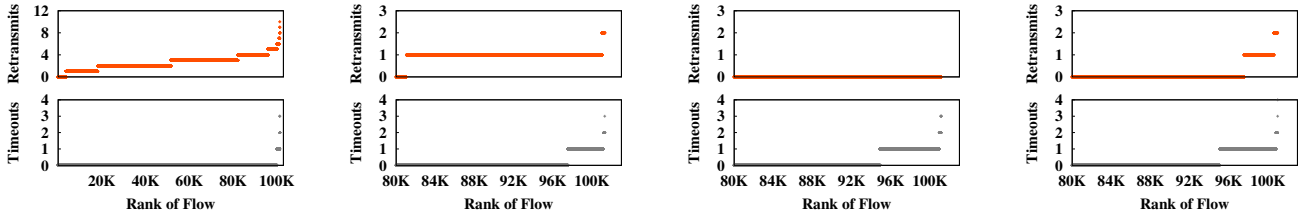
Table 3: MMPTCP vs TCP and PS (λ = 2560)

3.4. *Effects of Hotspots*

In this section we evaluate how each transport protocol reacts when hotspots appear in the network. These hotspots may occur for several reasons in modern data centres, including (1) contention between traffic flowing from the Internet to data centres, (2) hardware failures or cable faults, and (3) uneven load distribution in some servers. In order to model hotspots at the core layer, we modify the drop tail queue size of hotspot links from

Dupthresh Value	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
FCT (mean - ms)	158	138	127	120	116	113	112	113	113	114	115	116	116	116	116	117	117	117	116	116	116
FCT (StdDev)	103	99	93	100	92	96	97	98	93	102	104	102	106	104	106	107	106	109	104	96	100

Table 4: FCT of short flows for different *dupthresh* values.



(a) *dupthr=esh*= = 3 (b) *dupthresh* = 9 (c) *dupthresh* = 23 (d) Auto *dupthresh*

Figure 4: Timeouts and fast retransmissions for each individual short flow for various *dupthresh* values and our approach (auto *dupthresh*).

100 to 10 packets[9]. To select links under the hotspot, we select all the links of some randomly selected core switches. In this way, we can monitor the hotspot areas by simply monitoring each core switch under the hotspot. For this set of experiments, we range the number of switches where hotspots occur from 0% to 60% of the total number of core switches (we refer to the percentage of core switches where a hotspot occurs as 'hotspot degree'). We have used a 4:1 oversubscribed FatTree topology and the permutation traffic matrix (λ is 2560 for the arrival of short flows).

It is expected that by increasing the hotspot degree, the overall network utilisation will decrease and the overall short flow completion time and mean loss rate will increase for all transport protocols. Figure 3(a) shows the average goodput achieved by long flows for each different scenario and transport protocol combination for short and long flows (error bars denote the standard deviation). It is noticeable that TCP and PS (in the PS_{TCP}) achieve the worst overall goodput for long flows. MMPTCP performs the best. This observation also highlights the weakness of PS and the strength of MMPTCP in handling network congestion. The results in Figure 3(b) are in-line with the ones presented in Figure 3(a); the overall network utilisation in this experiment is directly related to the overall goodput of long flows because the majority of flows in the Permutation matrix cross the network core. Most of the data is generated by these long flows. Overall, as the number of hotspots increases, MMPTCP behaves well and consistently achieves a high mean core utilisation. Figure 3(c) illustrates the FCT of short flows. MPTCP with eight subflows performs the worst, being unstable for short flows, as denoted by the measured standard deviation. When the hotspot degree is 60%, the average FCT for TCP's short flows is 80.5ms with a standard deviation of 214ms. We observed that the second highest standard deviation is for TCP. This implies that hotspots heavily influence short flows when these are handled by TCP. In the previous subsection we showed the inability of TCP to use network resources efficiently for long flows, even without any hotspots, due to ECMP hash collisions.

Figure 3(d) illustrates the mean core loss rate achieved by each transport protocol. MMPTCP achieves the lowest mean loss rate for all hotspot degrees. By increasing the hotspot degree, the mean loss rate for PS_{TCP} and TCP increases significantly as, in both simulation scenarios, TCP handles short flows. The completely opposite result is observed for MMPTCP and PS because in both simulation scenarios PS is used for handling short flows. The intuition following this experiment is that the burstiness of data centre traffic, which arises from short TCP flows, is smoothed out with MMPTCP. In other words, TCP (for handling short flows) cannot handle congestion effectively. On the other hand, MMPTCP not only prevents congestion by scattering packets in the network, but also handles it effectively by shifting traffic away from congested areas, after switching to MPTCP (Phase 2). This is the main reason that MMPTCP achieves the lowest loss rate for all hotspot degrees, compared to other simulation scenarios. PS is not capable of dealing with hotspots effectively since it cannot detect them, as there is no notion of a flow (or subflow) that uses a specific network path where a hotspot appeared. Instead, packets are scattered in the network through all available paths (similar to VLB [18]).

3.5. MMPTCP and Duplicate ACK Threshold

In this section, we examine how the *dupthresh* value affects MMPTCP's performance during its first phase. We then evaluate the proposed approach for *preventing* spurious retransmissions due to packet reordering (see Section 2.4).

To explore the effect of packet reordering, we conducted a series of simulations with the *dupthresh* value ranging from 3 to 23[10]. We have simulated a FatTree topology with 128 servers running short and long MMPTCP flows. 33% of servers exchange data through long background flows and the remaining

[9]To select a size for the drop tail queue, we examined various queue sizes, ranging from 10 to 50 packets, and it turned out that 10 packets can best represent the behaviour of the studied transport protocols.

[10]Note that the dupack threshold value for MMPTCP's second phase is *3*, the default value for TCP and MPTCP. Packet reordering in a single path (for TCP or a single MPTCP subflow) is very unlikely in data centre networks (and non-existent in the simulated model)

67% of servers establish short flows (λ = 256). The results are shown in Table 4. It is clear that the default *dupthresh* value of three results in the worst average FCT (158ms). The observed average FCT decreases significantly as the *dupthresh* increases (e.g. when the *dupthresh* is nine). For values larger than nine, average FCTs remain unchanged while the standard deviation slightly increases.

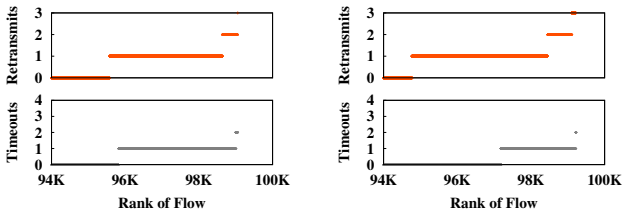
To get a better grasp of the problem, we look at the number of fast retransmissions and timeouts experienced by each short flow. Figure 4 shows the results for *dupthresh* values of 3, 23, 9 and *auto* (our approach). At the one extreme (*dupthresh* = 3), we observe the highest number of fast retransmissions (many of which are spurious) and the lowest number of timeouts (4(a)). At the other extreme (*dupthresh* = 23), there are no fast retransmissions, and the number of timeouts is the largest (4(c)). The best performance is observed when the threshold is set to nine (4(b)). For that threshold, the majority of flows were completed without fast retransmissions (~81000) or with only one (~19000); a few flows experienced two fast retransmissions.

The results of this experiment do not lead to any concrete value for the *dupthresh* since they are only valid for this particular network setup. By altering the network topology and/or traffic workload, the ideal *dupthresh* value would potentially be different. Figure 4(d) shows the results for the proposed approach of dynamically setting the threshold based on the location of the communicating hosts. Our approach significantly decreases the number of spurious retransmissions due to packet reordering by adjusting the value of *dupthresh* based on topology-specific information, as observed in Figures 4(b) and 4(d). Our approach only slightly increases the number of timeouts compared to *dupthresh* of 9[11]. Increasing or adjusting the *dupthresh* value is a challenging task as the TCP New Reno sender could lose its ACK clock, especially when the value of the *dupthresh* is larger than the congestion window (this condition is common under high loss rates). If any packet gets dropped in such scenarios, TCP needs to wait for a retransmission timeout to be triggered. For example, in the above simulations, 85% of network flows traverse the network core due to the used Permutation traffic matrix. This implies that with the dynamic approach a majority of short flows set their *dupthresh* values to 19. If any segment gets dropped at the first five RTTs, either at the beginning of data transmission or after any timeout event, the corresponding subflow needs to wait until a retransmission timeout is triggered. Therefore, the large *dupthresh* value is the main reason that auto *dupthresh* results in a slightly larger number of timeouts, compared to a *dupthresh* value of 9. In order to improve the performance of auto *dupthresh* and hence MMPTCP in such cases, we integrate the TCP Limited Transmit [24] mechanism in MMPTCP's first phase, as described in Section 2.4.

3.6. MMPTCP and Limited Transmit

We evaluate the performance of LT-enabled MMPTCP through a simulation scenario where both long and short flows

[11] We used this solution for all simulations conducted with MMPTCP and PS in this paper.



(a) MMPTCP (b) MMPTCP$_{LT}$

Figure 5: MMPTCP vs MMPTCP$_{LT}$: Timeouts and fast retransmissions

are transported by the LT-enabled version of MMPTCP (referred to as MMPTCP$_{LT}$). We compare its performance with the non-LT version of MMPTCP in a 2:1 oversubscribed FatTree topology with 256 servers using the Permutation traffic matrix (λ = 256). 53% (135) of servers send background flows and the remaining 47% (121) of servers send short flows. The results are shown in Table 5. MMPTCP$_{LT}$ greatly improves the mean FCT (and decreases the standard deviation) of short flows without reducing the overall network utilisation nor the goodput of long flows.

Transport Protocol	Short Flow Completion Time (mean/stdev)	Long Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
MMPTCP	98.9/±74.8 ms	72.9/±17.3 Mbps	72 %	0.0053 %
MMPTCP$_{LT}$	89.1/±67.2 ms	72.9/±18.0 Mbps	72 %	0.0051 %

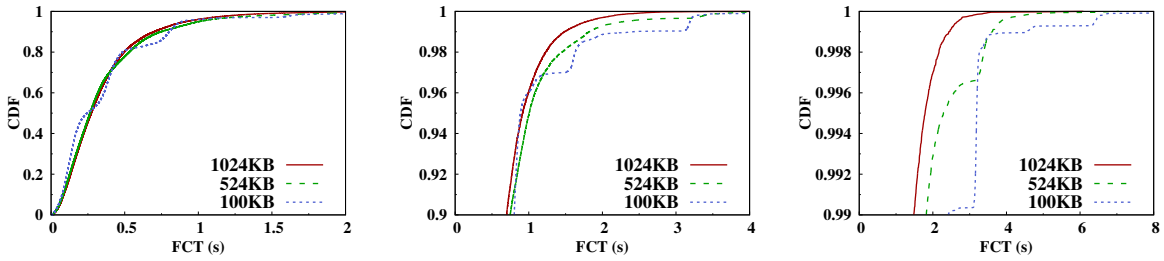
Table 5: MMPTCP vs MMPTCP$_{LT}$

We also look at the total number of fast retransmissions and timeouts for each individual short flow in Figure 5. MMPTCP$_{LT}$ results in slightly more fast retransmissions and fewer timeouts compared to MMPTCP. In fact, MMPTCP$_{LT}$ protects short flows from losing their ACK clocks when a high *dupthresh* value is used (e.g. 19). Therefore, the completion time of the majority of short flows is decreased when the LT-enabled version of MMPTCP is used.

3.7. MMPTCP and Switching Threshold

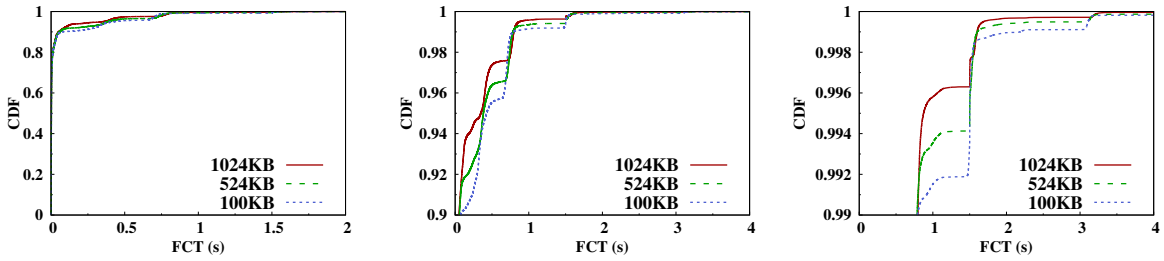
In this section we study how the switching threshold affects MMPTCP's performance, with respect to (1) the completion time of short flows when their size is lower or higher than the switching threshold; and (2) the goodput of long flows.

For the first experiment, we conducted two sets of simulations ranging the switching threshold from 100KB to 1024KB. In the first set, we select the size of short flow uniformly at random from the range of 1KB to 1024KB. In the second set, we select the size of all network flows from a real production data centre workload [17], in which 50% of all flows only contained 1 packet and 90% of all flows contained less than 267 packets. Figures 6 and 7 show the results for the first and the second set of simulations respectively. When the switching threshold is well below the maximum possible size of short flows, the FCT of some short flows (at the end of the tail) increases. This can be seen in Figure 6(c). However, as the switching threshold gets closer to 1024KB, the FCT of short flows at the tail of the distribution is decreased. One can also make similar observation in



(a) FCT of short flows (b) FCT of short flows > 90% (c) FCT of short flows > 99%

Figure 6: The effect of the switching threshold on short flows' completion time. The size of short flows is selected from the range of 1KB to 1MB.



(a) FCT of all flows (b) FCT of all flows > 90% (c) FCT of all flows > 99%

Figure 7: The effect of the switching threshold on flow completion time. The size of flows is retrieved from a production data centre workload [17]

Figure 7(c) where the longest flow has a size less than 3MB. If a flow's size is much greater than MMPTCP's switching threshold, MMPTCP may exhibit the similar problem as the standard MPTCP running multiple subflows. The switching threshold should therefore be set above the maximum expected size of a short flow.

Simulation Duration	Transport Protocol	Long Flow Goodput (mean / stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
1 Second	**MPTCP**	67.2/±14 Mbps	73.8 %	0.002 %
	MMPTCP	66.9/±14 Mbps	73.5 %	0.002 %
2 Seconds	**MPTCP**	75.5/±9.8 Mbps	82.0 %	0.003 %
	MMPTCP	75.2/±9.8 Mbps	81.6 %	0.003 %
5 Seconds	**MPTCP**	81.7/±7.5 Mbps	88.1 %	0.003 %
	MMPTCP	81.7/±7.4 Mbps	88.0 %	0.003 %
20 Seconds	**MPTCP**	86.1/±5.0 Mbps	92.6 %	0.003 %
	MMPTCP	86.1/±4.9 Mbps	92.6 %	0.003 %

Table 6: MMPTCP vs MPTCP

In order to describe the second experiment, let us assume that only long flows exist in the network. MMPTCP then switches to its second phase when the switching threshold is reached. At the time of switching, though, MMPTCP might have a very large window of data in flight. Unlike TCP, the congestion window here does not relate to the congestion state of a specific path, given that packets are scattered in the network during MMPTCP's first phase. After the threshold is reached, each newly established subflow ought to probe the network in order to prevent congestion collapse. In other words, newly established MPTCP subflows are not allowed to burst traffic into the network after switching because they assume that MMPTCP was sending aggressively before switching. But how does MMPTCP's switching threshold affect the goodput of long flows? To answer this question, we experiment with two simulation setups that only consist of long flows in a FatTree topology with 128 servers and the Stride matrix. For these two setups we use MMPTCP and MPTCP (with eight subflows) to transport all data for long flows, respectively. Examining the performance of these schemes in short time scales is important because the longer the simulations take to finish (and the long flows exist), the less difference in performance between the two protocols can be observed. For example, letting long flows to run for 20 seconds completely smooths out any performance glitches with respect to the switching threshold (which is 100KB in this case), as shown in Table 6. MMPTCP achieves almost identical results to MPTCP for all different long flow durations. We conclude that MMPTCP's switching mechanism effect on the goodput of long flows is negligible since newly established subflows (in phase 2) can fully utilise the access link capacity in a few RTTs.

3.8. MMPTCP and Incast

TCP Incast [29] occurs when a large number of synchronized short flows hit the same switch queue in the data centre. In partition/aggregate workloads Incast occurs at the queue of the switch port connected to the aggregator. According to [8], "Incast-like problems do happen in production environments and they matter - degrading both performance and, more importantly, user experience". Responses that are transported by flows that experience Incast will almost certainly miss the aggregator deadline and be left out of the final results [8]. In this section we study how MMPTCP behaves in simulated scenarios that trigger Incast, also in contrast to TCP, MPTCP and PS. To model Incast, we simulate a 1:1 oversubscribed FatTree

topology with 128 servers and initiate a number of parallel short flows (ranging from 20 to 100). The recipient of the data for all these flows is a single, randomly selected end-host in the simulated topology. Simultaneous and synchronised[12] short flows are scheduled every 500ms over the course of the simulation, hence Incast occurs periodically several times during each simulation run. There are no long flows in this experiment.

Table 7 shows the results. MPTCP with eight subflows performs the worst; compared to all other transport protocols, MPTCP completed the least number of short flows; a large number of connections could not be immediately established because SYN packets were being repeatedly dropped. Those connections that were established suffered from excessive timeouts; increasing the number of parallel flows made the problem worse. MMPTCP$_{LT}$ performs similarly to TCP. However, unlike MMPTCP$_{LT}$, MMPTCP performs slightly worse than TCP because it cannot prevent timeouts when packets get dropped and the *cwnd* value is smaller than the *dupthresh* value. In this experiment, MMPTCP behaves identically to PS[13] since all flows were short (with a size smaller than the MMPTCP switching threshold which was the default value of 70KB).

No. of Parallel Flows	Transport Protocol	No. of Completed Short Flows	Short Flow Completion Time		
			Mean/Stdev	Median	Upper Quartile
20	**MMPTCP**	16000	261.3/±81.0 ms	277.2 ms	288.5 ms
	MMPTCP$_{LT}$	16000	180.0/±38.9 ms	99.5 ms	108.7 ms
	TCP	16000	118.4/±59.6 ms	99.9 ms	109.4 ms
	MPTCP	7884	6.5/±5.5 s	6.22 s	6.26 s
40	**MMPTCP**	32000	416.9/±285 ms	452.7 ms	486.8 ms
	MMPTCP$_{LT}$	32000	186.0/±68.3 ms	165.2 ms	249.9 ms
	TCP	32000	203.4/±76.4 ms	169.4 ms	273.4 ms
	MPTCP	1975	7.6/±6.8 s	6.2 s	10.5 s
100	**MMPTCP**	47486	1.5/±3.4 s	767.4 ms	935.3 ms
	MMPTCP$_{LT}$	64825	2.2/±4.3 s	746.6 ms	1.6 s
	TCP	61110	2.1/±4.5 s	726.0 ms	1.0 s
	MPTCP	263	6.9/±7.8 s	4.5 s	10.5 s

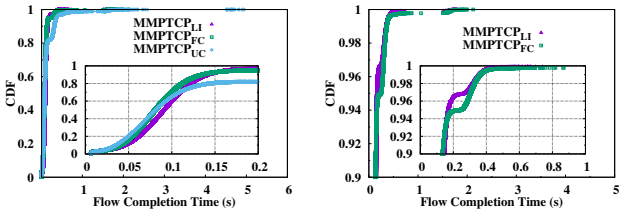
Table 7: Incast Evaluation

Simulation Name	Short Flow Completion Time (mean/stdev)	Long Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)	Access Layer Loss Rate (mean)
MMPTCP$_{FC}$	114/±127 ms	58.9/±18 Mbps	70.8%	0.014%	0.007%
MMPTCP$_{UC}$	179/±270 ms	50/±18.5 Mbps	63.8%	0.104%	0.041%
MMPTCP$_{LI}$	116/±101 ms	61.9/±20 Mbps	74.9%	0.007%	0.007%

Table 8: MMPTCP$_{FC}$, MMPTCP$_{UC}$ and MMPTCP$_{LI}$

3.9. MMPTCP and Multi-Path Congestion Control

In this section we evaluate MMPTCP's behaviour for different congestion control algorithms (*Fully Coupled (FC) [30]*, *Uncoupled-TCP (UC) [31, 32]* and *Linked Increases (LI)* [33]) during its second phase. We have used simulation scenarios similar to the ones presented in Section 3.4 and varied the congestion control algorithm used by MMPTCP's second



(a) CDF [0 - 1] (b) CDF [0.9 - 1]

Figure 8: Short Flow Completion Time (CDF) for MMPTCP$_{FC}$, MMPTCP$_{UC}$ and MMPTCP$_{LI}$

phase. The results are presented in Table 8. When *UC* is used, MMPTCP performs the worst with respect to short flow completion time (the observed standard deviation is also high). Additionally, with UC we observed the lowest overall goodput and network utilisation compared to the other algorithms. This is because the loss rates at the access and core layers are significantly increased compared to MMPTCP$_{LI}$ and MMPTCP$_{FC}$. The high loss rate is because eight subflows of MPTCP compete for the sender's access link capacity independently and aggressively, so the majority of packet loss occurs at that layer (the last column of Table 8 shows the mean loss rate at the access layer). As a result, the loss rate and network utilisation is lower at the core and aggregation layers.

The mean core loss rate for MMPTCP$_{FC}$ is double the rate of MMPTCP$_{LI}$. Network utilisation is also lower compared to MMPTCP$_{LI}$. The reason is that, with FC, traffic is shifted to the least congested paths, but those paths may become congested quickly since all short flows also use those paths, therefore traffic oscillates between available paths. This oscillation causes delays in some short flows. This reasoning can be confirmed by Figure 8 that illustrates the cumulative distribution function of short flow completion times. Flow completion times for MMPTCP with UC are heavy-tailed compared to MMPTCP with LI and FC. FC performs slightly worse compared to LI with respect to short flow completion time, especially at the tail (see Figure 8(b)).

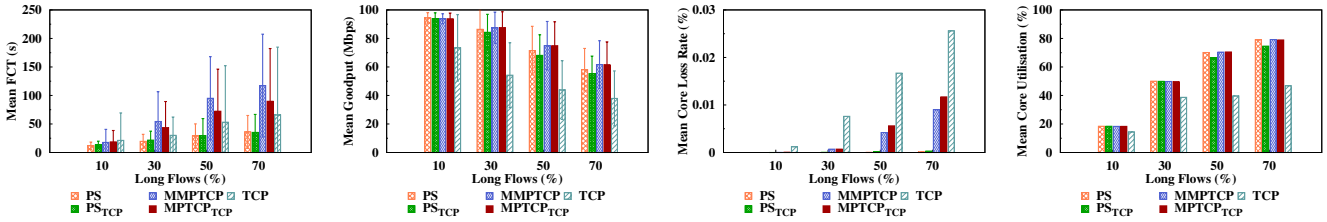
3.10. Influence of Network Load

In this section we study the behaviour of MMPTCP (also compared to TCP, PS and MPTCP as the network load increases. We simulate a 2:1 oversubscribed FatTree topology. To model the varying load, we range the percentage of servers running long flows from 10% to 70% of the total number of servers; i.e. the total network load increases with the number of long flows. Varying the percentage of servers running long flows does not affect the total number of generated short flows. For example, if 10% of servers run long flows then the remainder (90%) are only involved in sending short flows. For a specific inter-arrival rate this means that each server will serve fewer short flows over the course of the simulation.

Figures 9(a), 9(b), 9(c) and 9(d) depict the results for the short flow completion time, long flows' goodput, loss rate and utilisation at the network core, respectively. When both long and short flows are served by PS (depicted as PS in the figure), we observe the most stable results for the average flow

[12]This is the only set of experiments where flows are synchronised so that we can observe Incast. In all other experiments flows' arrival follows a Poisson process.

[13]For this reason PS does not appear in Table 7 given that its performance is identical to the performance of MMPTCP.



(a) Mean FCT of short flows (b) Mean goodput of long flows (c) Mean core loss rate (d) Mean core utilisation

Figure 9: Influence of Network Load

completion time for all studied network loads (Figure 9(a)); the mean and standard deviation increase slowly and consistently along with the network load. As mentioned before, this is an ideal scenario and performance diminishes dramatically when PS competes with TCP or MPTCP. When TCP is used for both long and short flows (depicted as TCP in the figure), the results are sensitive to the increasing load; the standard deviation increases significantly. As the network load increases, flow completion times for the MMPTCP scenario becomes slightly higher than the $\text{MPTCP}_{\text{TCP}}$ scenario. This is not surprising because in $\text{MPTCP}_{\text{TCP}}$, short flows are handled by TCP, and thus their packets are delivered from sender to receiver in-order. In fact, this explains why MMPTCP achieves slightly lower FCT compared to $\text{MPTCP}_{\text{TCP}}$ at 10% load.[14] Note that, as discussed earlier, modern data centres present extremely low packet drop probability, so the network load of 10% in this experiment is the most realistic one. According to [34, 17], long flows constitute only 10% of the total number of flows in data centres, although they carry the majority (90%) of the data. As a result, MMPTCP's loss in performance for heavier loads, which is due to packet reordering, is not a crucial flaw given its excellent performance for realistic workloads.

Figure 9(b) shows the overall goodput of long flows. It is expected that by increasing the number of long flows, the average goodput will decrease gradually for all protocol combinations, given that the network capacity is finite. We observe the worst goodput for long flows when both short and long flows are transported by TCP. When the load is low, all protocols (except TCP) perform equally well with respect to the goodput of long flows. At high loads, the observed mean goodput of long flows is the best for the MMPTCP and $\text{MPTCP}_{\text{TCP}}$ scenarios, where long flows are handled by MPTCP.

Figure 9(c) shows the mean loss rate at the network core for various network loads. When PS is used for handling long flows (in PS and PS_{TCP} scenarios), the mean loss rate is significantly lower, compared to all the other protocols. MMPTCP behaves better than the rest of the protocols.

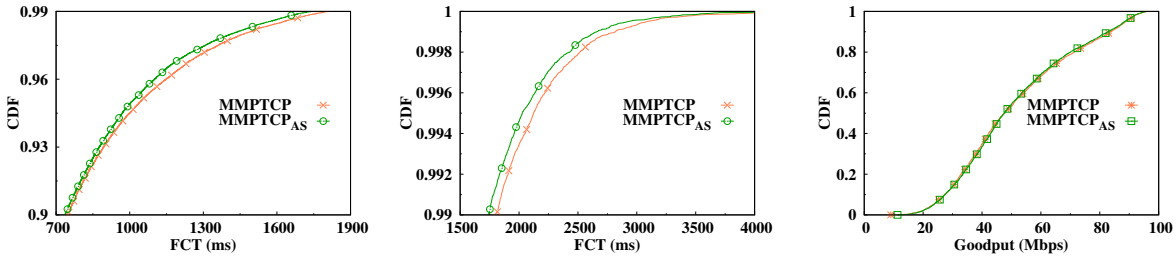
Figure 9(d) illustrates the mean utilisation at the network core. All protocols, except TCP, perform equally well in utilising network resources at the core layer. Note that with Permutation traffic matrix the majority of source hosts are connected to destination hosts that are located in different pods, so their flows cross the network core (see Section 3.1). This is the reason why the overall utilisation at the network core increases with the network load. ECMP collisions result in very poor performance for TCP.

3.11. Dynamic Selection of Subflow Number

In this section we examine the performance of MMPTCP when the number of active subflows during its second phase is calculated based on topology-specific information, as described in Section 2.5 (denoted as 'Auto Subflow' (AS) in the figures below). To evaluate this scheme, we use a 8-ary, 4:1 oversubscribed FatTree topology (consisting of 512 servers). Short flows are scheduled based on the Random traffic matrix, so that traffic crosses the access, aggregation and core network layers, which is important for evaluation the 'Auto Subflow' scheme. Based on the number of equal-cost paths in our network topology, if a flow crosses the core, aggregation and access layers, the sender MMPTCP host should open 8, 4, and 1 subflow(s) at the second phase of data transmission, respectively. Figure 10 shows the results. MMPTCP with the AS scheme enabled ($\text{MMPTCP}_{\text{AS}}$) results in lower FCT for short flows at the tail (Figure 10(a) and 10(b)) compared to standard MMPTCP. This improvement comes without any penalty regarding the goodput of long flows, which, as shown in Figure 10(c), is almost identical for both schemes.

We conclude that the 'Auto Subflow' scheme is beneficial to MMPTCP as well as to standard MPTCP because it effectively prevents needless consumption of valuable resources in the network and end-hosts, eliminating unnecessary subflows. As a result, AS improves the FCT of competing short flows; if a multipath flow competes with single path flows at a highly loaded bottleneck link at the access layer (e.g. the last mile link), then it may take eight times more bandwidth than the single path flows because it can send at least 16 packets on every RTT (two packets per each subflow) whereas a single path flow can send at least two packets on every RTT (because by default the minimum window size is set to two in Linux) [35]. This condition is particularly problematic in data centres because the bandwidth-delay product is typically small (e.g. 30 packets), so an MPTCP flow alone can take at least half of the available bandwidth of an access link. This problem becomes worse as the number of MPTCP flows sharing an access link increases.

[14] We refer the interested reader to [15] (Chapter 5, from p.122), which examines the performance of $\text{MPTCP}_{\text{TCP}}$ against MMPTCP in a wide network scenarios.



(a) FCT of short flows - 90th percentile (b) FCT of short flows - 99th percentile (c) Goodput of long flows

Figure 10: MMPTCP with Auto Subflow mechanism reduces the FCT for short flows, especially at the tail, compared to standard MMPTCP

4. Discussion and Future Work

During MMPTCP's first phase, the congestion window operates over multiple paths. When loss in one of the links in the network core occurs, a congestion signal is received and, in turn, the sender halves its congestion window. However, many other scattered packets may be crossing uncongested paths and therefore halving the congestion window is an overkill. On the other hand, if loss occurs at a bottleneck link at the access layer, then the reaction of TCP congestion control is correct. The research question here is how can we distinguish these two signals and react appropriately. Our hypothesis is that reacting to congestion proportionally to the extent of congestion will allow detection of these two signals. We thus expect that employing the DCTCP-like congestion control could be a viable solution for distinguishing these two signals. If a congestion signal comes from random links at the network core then the proportion of congestion signals, during one RTT, is very low so the sender should not reduce its sending rate. However, if it is from a bottleneck link at the access layer, DCTCP reacts similarly to TCP. Further investigating the potential benefits of a DCTCP-like approach for handling congestion during MMPTCP's first phase is part of our future work.

A brief discussion on the influence of the topology on the protocol behaviour can be found in [7]. Like MPTCP, the performance of MMPTCP is impacted by the network topology. The key factors are: (1) path diversity, (2) path length, and (3) link rate diversity.

Path diversity. If a network topology provides higher path diversity, MMPTCP can scatter packets across more paths, further reducing burstiness in traffic. However, exploiting a multipath transport protocol is generally less beneficial when a network (or parts of a network) does not provide multipath. This has been demonstrated in section 3.8 where both MMPTCP and MPTCP performed poorly under the incast-like conditions, mainly due to the lack of path diversity at the access layer of the network. Possible approaches to improve the MMPTCP performance under such conditions are to utilize a multi-homed network topology such as Dual-Homed FatTree (DHFT) [7], and/or equip MMPTCP with an ECN-capable multipath congestion control algorithm such as Adaptive MultiPath (AMP) [36] that performs as good as DCTCP under the incast-like conditions.

Path length. In this paper, we studied the performance of MMPTCP under a FatTree topology that provides equal-cost paths between all pairs of servers. This is not the case with a BCube [3] topology that follows the hypercube structure in which the length of paths between a pair of servers could be different. As the path length is directly influenced on the end-to-end latency, MMPTCP may face more episodes of packet reordering with a BCube-like topology compared to a FatTree topology. One possible approach to minimize the impact of packet reordering is to adjust the *dupthresh* value dynamically. We currently proposed to adjust the *dupthresh* value by exploiting the topology-specific information (see §2.4). A similar approach can be exploited for other topologies such as VL2.

Link rate diversity. A key difference between VL2 and FatTree topologies is that VL2 has 10 times faster links between its switches than its hosts and switches. Due to this difference we expect to see a better performance from MMPTCP in VL2 compared to FatTree. Our intuitions are twofold. Firstly, creating persistent congestion on a link at the core/aggregation layer of VL2 [2] requires at least 11 competing long flows originating from different hosts. With FatTree [1] this number is two due to the homogeneity of links. Secondly, transient congestion frequently occurs in VL2 due to traffic burstiness of short flows, and MMPTCP is designed to handle well such traffic patterns. In addition, the scattered packets at the initial phase of MMPTCP may experience queuing delays less frequently in VL2 compared to FatTree because the persistent congestion less frequently occurs in VL2."

Advanced QoS features have become increasingly available in data centre switches [37, 38]. Our hypothesis is that if packets of the initial phase of MMPTCP are marked as high priority and forwarded through different queues, then MMPTCP will be even more effective in helping latency-sensitive shortb flows to meet their deadlines. Evaluating such an approach is also part of our future work.

5. Conclusion

In this paper, we first showed through simulations that MPTCP is not efficient for short flows. We concluded that MPTCP is ill-suited to handle short flows because a fraction of them experience excessive timeouts. We proposed MMPTCP as a means to address this problem by combining packet scattering and multipath transport so that short flows can be efficiently

dealt with during the packet scattering phase. Our extensive experimental evaluation in simulated FatTree topologies showed that MMPTCP is practical and decreases flow completion time for short flows while retaining excellent goodput for large flows compared to MPTCP with a fixed number of subflows (eight in our simulations). We also observed that MMPTCP not only reacted to congestion gracefully but also prevented it to a great extent, therefore significantly decreasing the overall loss rate of all links in the network and increasing network resources' utilisation.

One of MMPTCP's challenges is to prevent, detect and react to spurious retransmission due to packet ordering during its first phase of data transport. We proposed a novel approach for preventing out-of-order packets which is to set the value of *dupthresh* based on topology-specific information. Our solution is based on the FatTree IP addressing scheme as it allows us to locate end-hosts according to their IP address. That is, the *dupthresh* is adjusted according to the destination IP address of a flow at connection establishment. Our evaluation showed that adjusting *dupthresh* in this way significantly prevents spurious retransmission. Our approach is also applicable for other data centre topologies, such as VL2.

MMPTCP switches to standard MPTCP after a specific volume of data has been transmitted to the receiver through packet scattering. We have showed that short flows' completion times are not significantly increased when the threshold is less than the size of the flow. However, our evaluation indicated that the threshold should be set to a value that is greater than the expected size of short flows (e.g. 1 to 10MB). We also showed that the switching mechanism does not affect the goodput of long flows because multipath transport can wrap up access link capacity in a few RTTs.

We conclude that MMPTCP is rapidly deployable in existing data centres as it can coexist with MPTCP and only requires existing data centre technologies such as ECMP. It can handle all network flows without high-level information from the application layer (e.g. flow sizes and deadlines). It decreases the bursty nature of data centres by exploiting equal-cost paths for delivering short flows. Finally, we showed that a topology-specific extension where the number of subflows opened during the second phase is dynamically set based on the location of the communicating hosts in the data centre can improve completion times of short flows without hurting the goodput of long flows.

[1] M. Al-Fares, A. Loukissas, A. Vahdat, A Scalable, Commodity Data Center Network Architecture, in: Proceedings of ACM SIGCOMM, 2008, pp. 63–74.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta, VL2: A Scalable and Flexible Data Center Network, Commun. ACM 54 (3) (2011) 95–104.

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers, in: Proceedings of ACM SIGCOMM, 2009, pp. 63–74.

[4] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, R. Govindan, Reducing Web Latency: The Virtue of Gentle Aggression, in: Proceedings of ACM SIGCOMM, 2013, pp. 159–170.

[5] A. Vulimiri, O. Michel, P. Godfrey, S. Shenker, More is less: reducing latency via redundancy, in: Proceedings of ACM HotNets, 2012.

[6] C. Hopps, Analysis of an Equal-Cost Multi-Path Algorithm, RFC 3782 (2004).

[7] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving Datacenter Performance and Robustness with Multipath TCP, in: Proceedings of ACM SIGCOMM, 2011.

[8] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data Center TCP (DCTCP), in: Proceedings of ACM SIGCOMM, 2010, pp. 63–74.

[9] B. Vamanan, J. Hasan, T. Vijaykumar, Deadline-aware Datacenter TCP (D2TCP), in: Proceedings of ACM SIGCOMM, 2012, pp. 115–126.

[10] C. Wilson, H. Ballani, T. Karagiannis, A. Rowtron, Better Never Than Late: Meeting Deadlines in Datacenter Networks, in: Proceedings of ACM SIGCOMM, 2011, pp. 50–61.

[11] A. Ford, C. Raiciu, M. Handley, S. Barré, J. Iyengar, TCP Extension for Multipath Operation with Multiple Addresses, RFC 6824 (2013).

[12] M. Kheirkhah, I. Wakeman, G. Parisis, Multipath-TCP in ns-3, CoRR. URL `http://arxiv.org/abs/1510.07721`

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: Dynamic Flow Scheduling for Data Center Networks, in: Proceedings of USENIX NSDI, 2010.

[14] M. Kheirkhah, I. Wakeman, G. Parisis, MMPTCP: A multipath transport protocol for data centers, in: Proceedings of IEEE INFOCOM, 2016.

[15] M. Kheirkhah Sabetghadam, MMPTCP: A Novel Transport Protocol for Data Centre Networks, Ph.D. thesis, University of Sussex (2016).

[16] A. Dixit, P. Prakash, Y. C. Hu, R. R. Kompella, On the impact of packet spraying in data center networks, in: Proceedings of IEEE INFOCOM, IEEE, 2013, pp. 2130–2138.

[17] T. Benson, A. Akella, D. A. Maltz, Network Traffic Characteristics of Data Centers in the Wild, in: Proceedings of ACM SIGCOMM, 2010, pp. 267–280.

[18] L. Valiant, A Scheme for Fast Parallel Communication, SIAM journal on computing 11 (2) (1982) 350–361.

[19] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883 (2000).

[20] J. Mahdavi, M. Mathis, S. Floyd, A. Romanow, TCP selective acknowledgment options, RFC 2018 (1996).

[21] S. Floyd, T. Henderson, The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 6582 (2002).

[22] M. Zhang, B. Karp, S. Floyd, L. Peterson, RR-TCP: A Reordering-Robust TCP with DSACK, in: Proceedings of IEEE ICNP, 2003.

[23] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, V. Kurien, Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis, in: Proceedings of ACM SIGCOMM, 2015, pp. 139–152.

[24] M. Allman, H. Balakrishnan, S. Floyd, Enhancing TCP's Loss Recovery Using Limited Transmit, RFC 3042 (2001).

[25] C. Hopps, Analysis of an Equal-Cost Multi-Path Algorithm, Tech. Rep. 2992 (November 2000).

[26] P. Costa, H. Ballani, K. Razavi, I. Kash, R2C2: A Network Stack for Rack-scale Computers, in: Proceedings of ACM SIGCOMM, 2015, pp. 551–564.

[27] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, G. Varghese, Conga: Distributed congestion-aware load balancing for datacenters, in: Proceedings of ACM SIGCOMM, 2014.

[28] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, M. Schapira, PCC: Re-architecting congestion control for consistent high performance, in: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 2015.

[29] Y. Chen, R. Griffith, J. Liu, R. H. Katz, A. D. Joseph, Understanding TCP Incast Throughput Collapse in Datacenter Networks, in: Proceedings of ACM WREN, WREN '09, 2009, pp. 73–82.

[30] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, D. Towsley, Multi-path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet, IEEE/ACM Trans. Netw. 14 (6) (2006) 1260–1271.

[31] K. Rojviboonchai, H. Aida, An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes, IEICE transactions on communications 87 (9) (2004) 2699–2707.

[32] H.-Y. Hsieh, R. Sivakumar, A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts, in: Proceedings of

MobiCom, 2002, pp. 83–94.
[33] C. Raiciu, M. Handley, D. Wischik, Coupled Congestion Control for Multipath Transport Protocols, RFC 6356 (2011).
[34] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The Nature of Data Center Traffic: Measurements & Analysis, in: Proceedings of ACM IMC, 2009, pp. 202–208.
[35] M. Kheirkhah, M. Lee, AMP: A Better Multipath TCP for Data Center Networks, CoRR abs/1707.00322. arXiv:1707.00322.
URL http://arxiv.org/abs/1707.00322
[36] M. Kheirkhah, M. Lee, AMP: An Adaptive Multipath TCP for Data Center Networks, in: Proceedings of IFIP/IEEE Networking, 2019.
[37] D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz, Detail: Reducing the flow completion time tail in datacenter networks, in: Proceedings of ACM SIGCOMM, 2012, pp. 139–150.
[38] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, pFabric: Minimal Near-optimal Datacenter Transport, in: Proceedings of ACM SIGCOMM 2013, 2013, pp. 435–446.